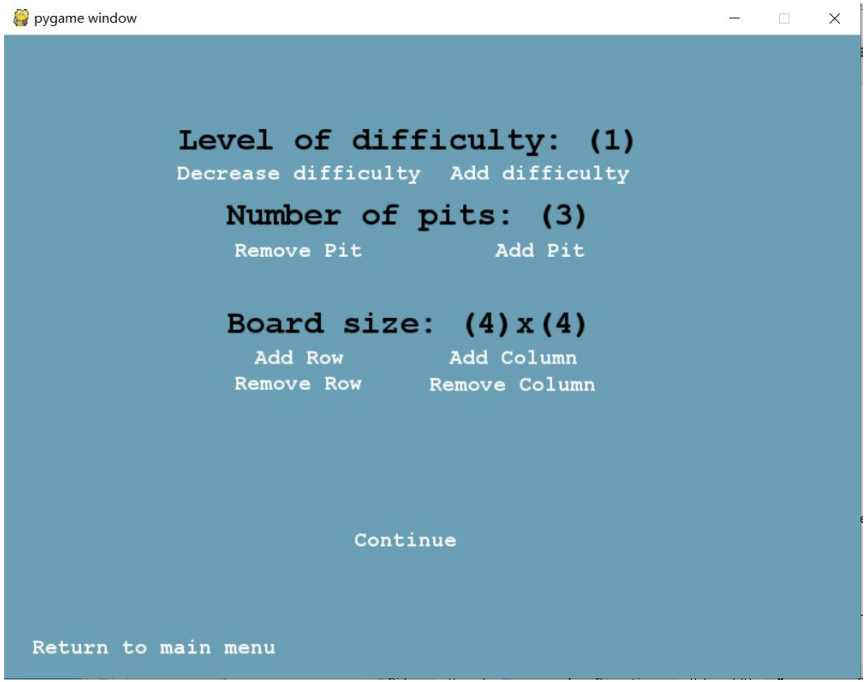
Welcome to the Wumpus World

Instructor:
Bram Van Heuveln
Wumpus World Generator:
Mark,Song,Alayn,Jessica,Samantha, Ian
Wumpus World Simulator:
Mitesh, Mindy, Xinyan, Qiao, Emma, Kevin

Intro to the UI windows



Open the gui3.py and enter the game you will see this page.
On this page you can set the level of difficulty and the Map information for manual game and single agent game.
There are four levels of difficulty. 0 and 1 are the levels that you don't need to shoot the wumpus; 2 and 3 are the levels that you have to shoot the wumpus.
There are some combination of level and pits that cannot generate a map and cause the program to crush, just try more reasonable combinations when it happens.

Intro to the UI windows

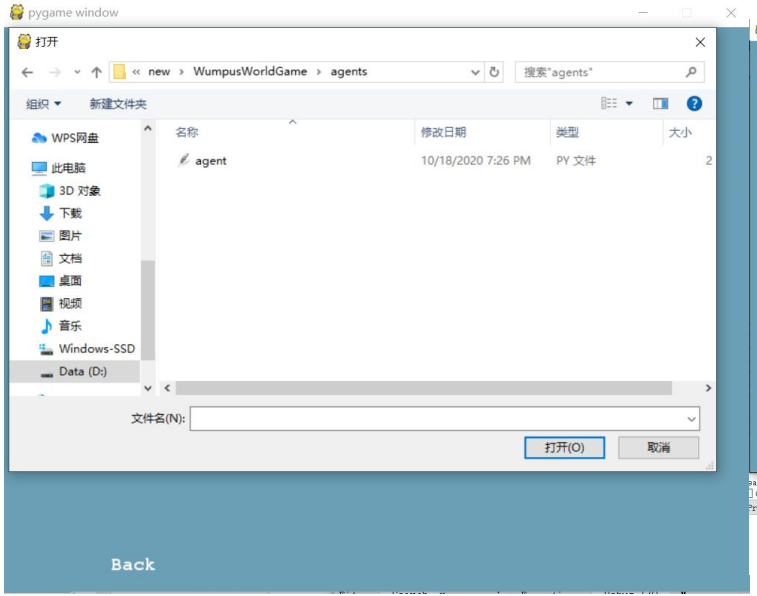
There are four modes to choose.

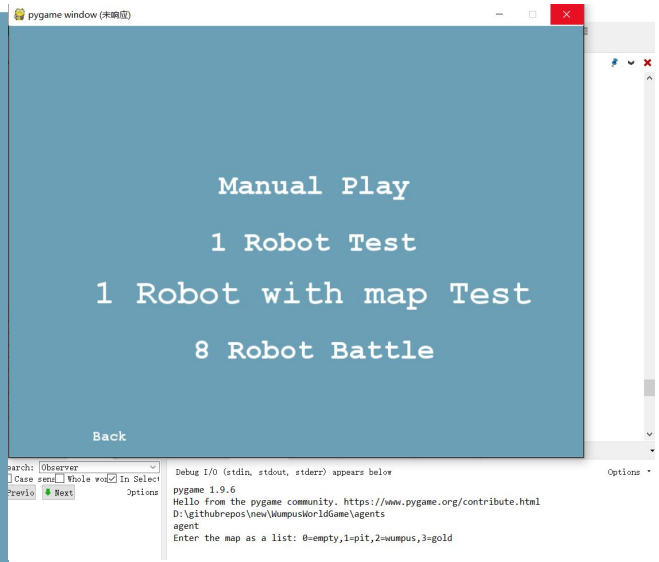
The Manual Play will start a game that you can play munally.

The 1 robot test will pop a window and ask you to choose your agent file and then let you play.

The 1 robot with map test will also ask you to enter a map in the command line.

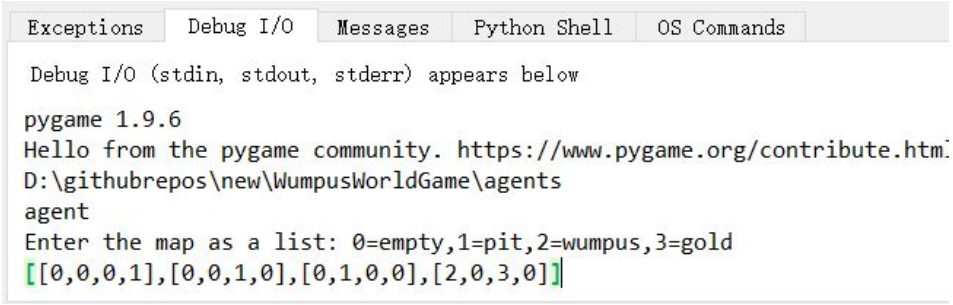
Intro to the UI windows





As you can see on the second picture, it ask you for a map.

Intro to the UI windows

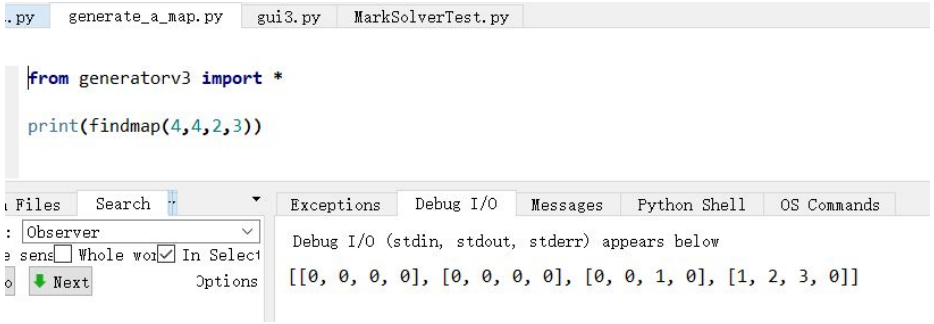


Enter a map like this will let you start the game.
The robot will start at (0,0) of the map. Below is the representation rule of the map:

```
board[i][j]'s up and down, left and right will be like:
                                    i=2  *    *    *
                                    i=1  *    *    *
                                    i=0  *    *    *
                                        j=0 j=1 j=2
And the robot will always start at point (0,0).
```

Intro to the UI windows

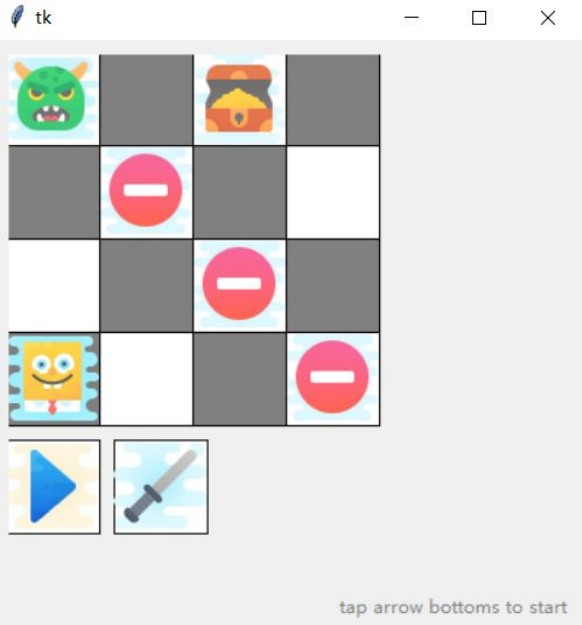
You can run generate_a_map.py to generate a random map.



The findmap function's input is:
findmap(sizex,sizey,pits,difficulty)

Some maps that I used to test my robot:
[[0,0,0,0],[0,0,0,0],[0,0,0,3],[0,0,0,0]]
[[0,0,2,0],[0,0,0,3],[1,0,0,0],[0,0,0,0]]
[[0,0,0,1],[0,0,2,3],[1,1,0,0],[0,0,0,0]]
[[0,0,0,1],[0,0,1,0],[0,1,0,0],[2,0,3,0]]

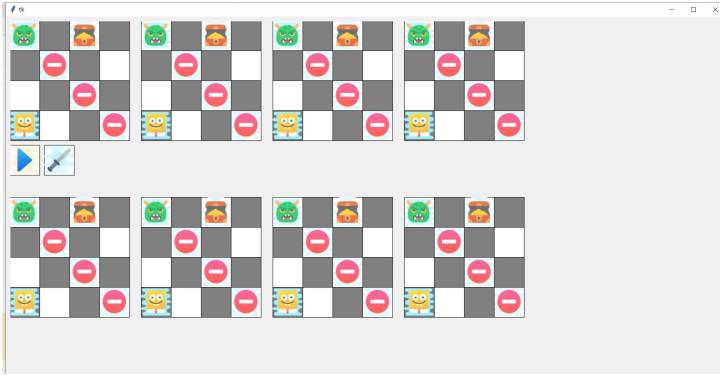
Intro to the UI windows



After entering the map, this window will pop up.
The blue button will let the robot automatically step through the whole game.
The knife button will let the robot make 1 step.

Intro to the UI windows

The 8 robot will ask you to choose 8 robots 1 by 1. Then it will ask you to enter a map on the command prompt. And then you will have 8 robots on screen like this:



After the game it will print out the result in the command line like below.
You can enter anything and the game will restart on the same map.

```
Debug I/O (stdin, stdout, stderr) appears below
00000 00000 00000 00000
00000 00000 00000 00000
00000 00000 here! 00000

FAIL
PIT
Agent 1 lose
Agent 2 lose
Agent 3 lose
Agent 4 lose
Agent 5 lose
Agent 6 lose
Agent 7 lose
Agent 8 lose
finished
```

If you cannot use the gui3.py:

limitedUI.py is based on tkinter only.

NoUIgame.py is a text based game.

These two are different versions of 1 robot with map test.

They assume you will put your agent.py under agents folder.

Enter the agent file name without the .py when it asks you for agent.

Enter the map like before as it asks you for the map.

How to write a basic robot

The agent.py shows you the structure of a robot that will always move right.

agent.py | generate_a_map.py | gui3.py | MarkSolverTest.py

Agent | __init__

```
class Agent:
    def __init__(self,sizex, sizey):
        ##sizex and sizey will give your agent the size of the map
        self.sizex=sizex
        self.sizey=sizey
        ##TODO: Put the variables you need for your agents here.


    ##TODO: define the functions you need here


    '''
    move(state) will read in the message from the game and return the move the agent will make based on the current information.
    This is the only function that will be called by the game and the name, param and return must not be changed.
    @param state will be a tuple (messages, 0)      0 is useless here.
           If you use a board which a list(list(set)) where the set keeps all the information about a node on the map,
           board[i][j]'s up and down, left and right will be like:
                                                  i=2  *   *   *
                                                  i=1  *   *   *
                                                  i=0  *   *   *
                                                      j=0 j=1 j=2
           And the robot will always start at point (0,0).
           The state[0]: messages will be a list of strings which might include: "CONTINUE", "BREEZE", "STENCH", "GLITTER", "KILLED-WUMPUS","GOLD" .
    @return This function should return a string "move_up", "move_down" , "move_left", "move_right" , "shoot_up", "shoot_down", "shoot_right", "shoot_left" based on the current state.
    '''
    def move(self,state):
        ##TODO: Implement your algorithm here
        return "move_right"
```

How to write a basic robot

In the game, only the move() function will be called by the game environment and it should return a string like: "move_up", "move_down" , "move_left", "move_right" , "shoot_up", "shoot_down", "shoot_right", "shoot_left".

You can write your functions to choose from the strings above to return and the robot will move accordingly.

How to write a advance robot

My suggestion is to break it down to at least 3 parts.

The first part is to generate the map according to the state that you received. Like a memory.

The second part is to reason through your current map in memory and see which spots can be visited but have not been visited yet.

The third part is to find a safe path from your current spot to a spot form the second part and let the robot follow that path.(or you have found the wumpus and decide to kill it, or you have the gold and decide to go back..)

Some other suggestions:

To create your map in memory, you might find the nodeclass.py helpful, you can store the map informations in list(list(node)) structure.

At some point you might want to step more than one step. To do that, you can use this structure in A_structure_you_can_follow.py:

```
def __init__(self,sizex,sizey):
    ##TODO: Put the variables you need for your agents here.
    self.board=[]
    self.nextmoves=deque()
    self.sizex=sizex
    self.sizey=sizey
    self.x=0
    self.y=0
def move(self,state):
    ##TODO: Implement your algorithm here
    if len(self.nextmoves)!=0:
        return self.nextmoves.popleft()
    self.parsemessage(state[0])


    if self.ivegotyouinmysight():
        self.itshighnoon()
        return self.nextmoves.popleft()

    if self.checkgold():
        self.tovictory()
        return self.nextmoves.popleft()

    nextx,nexty=self.checknextpos()
    x=self.x
    y=self.y
    history=set()
    path=deque()
    thepath=self.dfs(x,y,nextx,nexty,history,path)
    self.nextmoves=thepath
    self.x=nextx
    self.y=nexty
    print(self.x)
    print(self.y)
    return self.nextmoves.popleft()
```

Thank you

Instructor:
Bram Van Heuveln
Wumpus World Generator:
Mark,Song,Alayn,Jessica,Samantha, Ian
Wumpus World Simulator:
Mitesh, Mindy, Xinyan, Qiao, Emma, Kevin